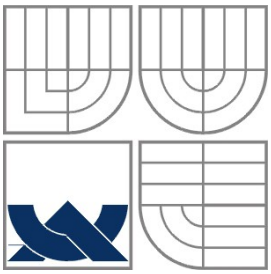


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ON-LINE POČÍTAČOVÉ HRY

ON-LINE COMPUTER GAMES

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE **ZDENĚK ŽAMBERSKÝ**
AUTHOR

VEDOUCÍ PRÁCE **ING. JAN ROUPEC, PH.D.**
SUPERVISOR

BRNO 2013

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2012/13

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Zdeněk Žamberský

který/která studuje v **bakalářském studijním programu**

obor: **Strojní inženýrství (2301R016)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

On-line počítačové hry

v anglickém jazyce:

On-line computer games

Stručná charakteristika problematiky úkolu:

Cílem práce je zvládnout technologii tvorby on-line počítačových her. Autor v rámci práce vytvoří ukázkové hry, které budou vhodné k propagaci studia na ÚAI.

Cíle bakalářské práce:

Autor se seznámí s technologií tvorby on-line počítačových her. Následně vytvoří ukázkové aplikace, které budou vhodné pro propagaci studia na ÚAI.

Seznam odborné literatury:

1. Anyuru A.: Professional WebGL Programming: Developing 3D Graphics for the Web. Wrox, 2012.
2. Seidelin J.: HTML5 Games: Creating Fun with HTML5, CSS3, and WebGL. Wiley, 2011.

Vedoucí bakalářské práce: Ing. Jan Roupec, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2012/13.

V Brně, dne 20.3.2013

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ
DĚKANÁT

Ing. Jan Roupec, Ph.D.
Ředitel ústavu

prof. RNDr. Miroslav Doupovec, CSc., dr. h. c.
Děkan

ABSTRAKT

Tato bakalářská práce se zabývá on-line počítačovými hrami, přesněji webovými hrami. První část je věnována popisu jednotlivých technologií, použitelných pro tvorbu webových her. Další 2 kapitoly jsou věnovány ukázkovým aplikacím, vytvořeným v rámci bakalářské práce.

ABSTRACT

This bachelor's thesis deals with on-line computer games, more precisely web games. First chapter is focused on technologies usable for web game development. Other two chapters are devoted to example aplications, develped as part of this bachelor's thesis.

KLÍČOVÁ SLOVA

On-line počítačové hry, web, webgl, svg, simulace tkanin, fyzikální simulace

KEYWORDS

On-line computer games, web, webgl, svg, cloth simulation, physics simulation

PROHLÁŠENÍ O ORIGINALITĚ

Prohlašuji, že práci jsem vytvořil samostatně, pod vedením Ing. Jana Roupce, Ph.D. a za použití citované literatury.

BIBLIOGRAFICKÁ CITACE

ŽAMBERSKÝ, Z. *On-line počítačové hry*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 41 s. Vedoucí bakalářské práce Ing. Jan Roupec, Ph.D..

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat svému vedoucímu bakalářské práce panu Ing. Janu Roupci, Ph.D. za rady týkající se tvorby bakalářské práce.

Obsah:

1 ÚVOD

Tato práce se zabývá on-line počítačovými hrami, v rámci kterých se zaměřuje na webové hry. Tedy hry spouštěné a zpravidla i běžící v okně webového prohlížeče.

V rámci práce byl proveden průzkum v oblasti jednotlivých technologií. Výsledkům je věnována jedna kapitola, která rozebírá jednotlivé technologie, použitelné pro tvorbu webových her.

Dale byly vytvořeny 2 demonstrační aplikace, které se snaží přiblížit možnosti vybraných technologií.

Cílem první aplikace bylo vytvoření aplikace s využitím standardu pro 3D grafiku WebGL, která bude provádět numerickou simulaci vlajky ve větru v reálném čase. Jejím cílem je také mimo jiné demonstrovat využitelnost WebGL, nejen pro zobrazování 3D grafiky, ale také k provádění výpočtů na grafické kartě, k dosažení vyššího výpočetního výkonu.

Cílem druhé aplikace bylo vytvořit aplikaci ve vektorovém grafickém formátu SVG, doplněném o skripty psanými v javaScriptu. Aplikace je založena na jednoduchém 2D fyzikálním enginu, který byl pro ni v rámci práce vytvořen. Aplikace má také za cíl načíst veškeré informace o „herním světě“ včetně poloh překážek z vhodně strukturovaného SVG souboru, a tím umožnit editaci herní scény z vektorového editoru, bez nutnosti zasahovat do skriptů.

2 TECHNOLOGIE PRO TVORBU WEBOVÝCH HER

Tato kapitola rozebírá jednotlivé technologie pro tvorbu webových her. Jsou zde shrnuty možnosti, výhody a nevýhody jednotlivých technologií. Zvýšená pozornost je věnována technologiím použitým při tvorbě ukázkových aplikací.

2.1 ECMAScript

2.1.1 JavaScript v prohlížečích

JavaScript je základní skriptovací jazyk používaný při tvorbě interaktivních webových stránek. JavaScript byl standardizovaný pod označením ECMAScript v roce 1997 [2] a dnes jej podporují všechny majoritní prohlížeče.

Z hlediska webových her byly dříve možnosti omezeny na manipulaci HTML elementy a jejich atributy a použití se omezovalo spíše na aplikace formulářového charakteru. Prohlížeče navíc používaly jednoduché interpretery, u kterých nebyla rychlost jeho vykonávání nijak vysoká. V posledních letech se zde však objevila snaha tvůrců prohlížečů JavaScript urychlit, javaScriptové enginy se staly složitějšími a začaly používat pokročilé techniky jako např. JIT (just-in-time), tj. kompilaci do nativního kódu procesoru za běhu. Tím došlo k mnohonásobnému urychlení [3][4]. S příchodem HTML5 pak přišlo rozšíření API a vznik nových elementů, umožnujících nízkoúrovňové grafické operace a přehrávání multimédií. Přibyla také možnost vícevláknových aplikací. JavaScript se tak v HTML5 stává velmi mocným nástrojem pro tvorbu webových her.

2.1.2 ECMAScript a jeho použití ve webových technologiích a jazycích

Použití ECMAScriptu se však neomezuje na využití jakožto JavaScriptu v HTML. V rámci různých dialektů někdy pod jinými názvy se objevuje i v jiných webových technologiích. Zde je krátký výčet technologií a jazyků použitelných pro tvorbu webových her využívajících různé formy ECMAScriptu s příslušnými názvy:

- HTML (JavaScript),
- SVG (JavaScript nebo ECMAScript),
- Adobe Flash (ActionScript),
- Silverlight (JavaScript) – zde však není primární jazyk.

2.2 HTML5

Přesto, že HTML je pouze značkovací jazyk (markup language) a nejedná se tedy přímo o programovací jazyk, ve kterém by bylo možné psát aplikace, poskytuje elementy, které lze v aplikacích využívat.

2.2.1 Vznik

HTML5 představuje v tomto ohledu určitou revoluci. Za jeho vznikem stojí WHATWG (Web Hypertext Application Technology Working Group), jež bylo vytvořeno jednotlivci ze společností zabývajících se vývojem prohlížečů [7], jako reakce na pomalý vývoj HTML. Čtvrtá verze byla totiž vyvinuta již na přelomu tisíciletí organizací W3C (World Wide Web Consortium), zabývající se do této verze vývojem standardu. Tato verze se však začala stávat nedostatečnou, a to hlavně v oblasti interaktivního a multimediálního obsahu a tvůrci stránek byli nuceni používat technologie třetích stran (např. Adobe Flash a jiné). Do standardu HTML5, na kterém nyní pracují WHATWG a W3C společně, byla proto přidána celá řada elementů, které mají toto řešit. Jako skriptovací jazyk je použit JavaScript jehož API bylo také rozšířeno.

2.2.2 Nové elementy

Z hlediska tvorby webových her a multimediálních aplikací jsou zajímavé především tyto elementy:

- **canvas** – je element do kterého lze kreslit. To je realizováno získáním kontextu (context) a to buď WebGL nebo 2d. Kontext 2d poskytuje plnohodnotné API pro kreslení vektorové grafiky tj. cest (paths), textu i rastrových obrázků. Při tvorbě cest lze využít úseček, kruhových oblouků a kvadratických/kubických bezierových křivek. API poskytuje funkce jako vypňování (filling), kreslení obrysu (stroking) a to i s použití určitého vzoru (dashing). Cestami lze také ořezávat (clipping). Scéna je kreslena sekvencí funkcí v JavaScriptu (není tvořena stromem elementů jak je tomu např. u SVG).
- **audio, video** – jsou elementy pomocí kterých je možno do stránky vkládat multimediální obsah. Tyto element můžou mít více zdrojů a tím umožnit alternativní soubor v případě, že prohlížeč nepodporuje určitý formát. Při přehrávání těmito elementy lze využít, jak jednoduché nastavení s použitím atributů popřípadě ovládacích prvků poskytnutých prohlížečem, tak nízkoúrovňového ovládání pomocí JavaScriptu.
- **SVG** – je element představující vektorovou grafiku ve formátu SVG. V HTML5 lze nově vkládat SVG tag přímo do html kódu, není už nutno používat další elementy jako např. embed nebo object.

2.2.3 Výhody

Mezi výhody použití HTML5 patří:

- nativní podpora prohlížečů – není nutné instalovat zásuvné moduly (pluginy),
- nízkoúrovňová grafika – to jak vektorová, tak 3D skrze WebGL,
- podpora multimediálního obsahu skrze elementy audio a video,
- hry a aplikace vytvořené pomocí nových elementů mohou snadno interagovat se zbytkem stránky,
- dobrá podpora – všechny majoritní prohlížeče podporují HTML5.

2.2.4 Nevýhody

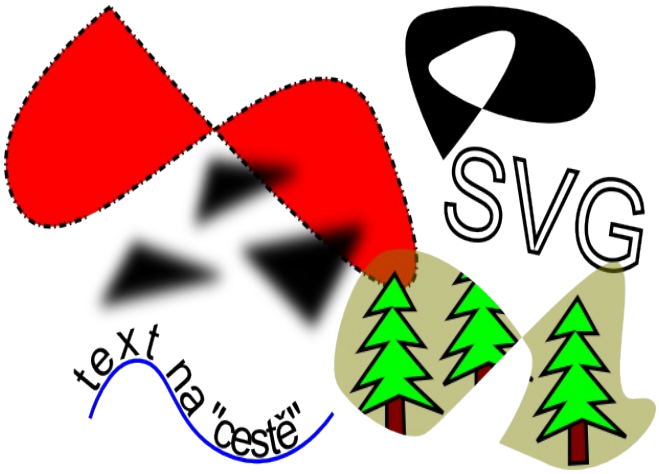
Nevýhody použití HTML5 jsou:

- zdrojové soubory v JavaScriptu jsou snadno získatelné uživatelem,
- velikost – zdrojové soubory nejsou nijak komprimovány,
- nový standard – podpora prohlížečů nemusí být úplná, může působit problémy u starších verzí prohlížečů.

2.3 SVG

SVG (Scalable vector graphics) je, jak název napovídá vektorový grafický formát, avšak i něm jdou tvořit webové hry. Umožňuje totiž skriptování s pomocí JavaScriptu (ECMAScriptu), a to buď přímo v souboru SVG nebo odkazem na externí soubor.

SVG je otevřený formát, jehož tvůrcem je konsorcium W3C. Je založen na formátu XML, tvoří jej tedy strom elementů. Pomocí těchto elementů lze tvořit základní geometrické tvary, cesty, text a vkládat bitmapovou grafiku. Lze jej vytvořit ručně pomocí textového editoru a nebo využít vektorový editor. K manipulaci s elementy SVG dokumentu se používá stejné API jako pro manipulaci s elementy HTML, tj. DOM (Document object model) API, samozřejmě doplněné pro potřeby SVG. Toto je nesporně výhoda pro toho, kdo již používal JavaScript v HTML. Další výhoda je, že soubor SVG s celou herní scénou lze vytvořit v libovolném vektorovém editoru, který pak stačí doplnit příslušnými skripty. Na Obr. 1 na které lze vidět demonstraci možností SVG a to včetně pokročilých efektů jako blur.



Obr. 1 Demonstrace možností SVG

2.3.1 Výhody

Mezi výhody SVG patří:

- široké možnosti na poli vektorové grafiky,
- nativní podpora prohlížečů – nevyžaduje zásuvný modul,
- využívá DOM API – manipulace s elementy pomocí JavaScriptu je podobná jak je tomu u html,
- snadná čitelnost – SVG je založen na XML,
- možnost využití řady vektorových editorů,
- možnost komprese (přípona svgz).

2.3.2 Nevýhody

Nevýhody SVG jsou:

- použitelnost pouze pro aplikace založené na vektorové grafice,
- scéna tvořená stromem elementů neposkytuje takovou flexibilitu, jak je tomu při kreslení sekvencí funkcí.

2.4 WebGL

WebGL je rozšíření standardu HTML5 [9], vycházející z OpenGL ES 2.0 [8], což je API a standard pro 3D grafiku na mobilních zařízeních, jež vychází z OpenGL 2.0. Autorem všech těchto API je neziskové průmyslové konzorcium Khronos Group. Hlavním rozdílem mezi OpenGL (ES) 2.0 a WebGL je, že zatímco OpenGL (ES) je API pro jazyk C, které je většinou implementováno přímo v rámci ovladače grafické karty, WebGL zprostředkovává jeho funkcionalitu v JavaScriptu a umožňuje vykreslování do HTML5 elementu canvas, jak bylo řečeno výše. Díky podobnosti s OpenGL by zvládnutí WebGL nemělo představovat problém pro člověka, který již v OpenGL programoval.

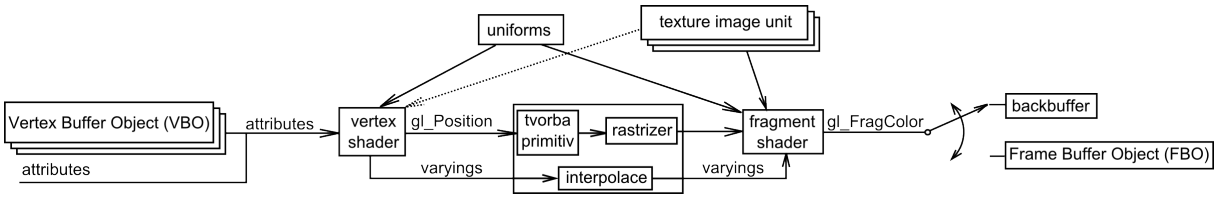
2.4.1 Způsob vykreslování

Vykreslování WebGL probíhá prostřednictvím tzv. pipeline, což je sekvence určitých na sobě závislých operací, kdy výstup jedné operace je vstup další. Operace, které tvoří WebGL pipeline lze vidět na Obr. 2 , z nichž jsou programovatelné vertex shader a fragment shader.

Shadery jsou v podstatě krátké programy (i když označení programy muže být matoucí viz dále), které jsou programovány pomocí jazyka GLSL ES (tj. OpenGL shading language pro mobilní zařízení). Tento jazyk vychází z jazyka C a podporuje skalární i vektorové datové typy. Aby bylo možno WebGL používat, je potřeba tyto shadery dodat. Ty se následně přeloží do strojového kódu

grafické karty. Poté je nutné vytvořit programy jimž se přiřadí dvojice vertex a fragment shaderu a provede se linkování. Takto vytvořenému programu lze již konfigurovat vstupy v podobě přiřazení do speciálních proměnných příslušnými funkcemi.

Vykreslování ve WebGL může mít dva různé cíle. První je framebuffer vytvořený v rámci elementu canvas. Ten se skládá se z dvou bufferů. Frontbufferu a backbufferu, přičemž frontbuffer je zobrazen a do backbufferu se vykresluje. Po ukončení vykreslování se oba buffery prohodí (toto kontroluje prohlížeč automaticky). Druhá možnost je kreslit do tzv. Frame Buffer Objectu (FBO). Tímto se provádí vykreslování mimo obrazovku, kdy se vykresluje do textury připojené k FBO. V obou případech se vykresluje do obdélníkové oblasti nastavitelné pomocí tzv. viewportu.



Obr. 2 WebGL pipeline (s názvy v angličtině)

Přesto, že mezi jednotlivými operacemi tvořícími pipeline je sekvenční vazba, uvnitř operací samotných takové vazby záměrně nejsou a na dnešních grafických kartách se vykonávají na velkém množství dat paralelně (v případě shaderů na mnoha shader procesorech současně).

2.4.2 Vertex shader

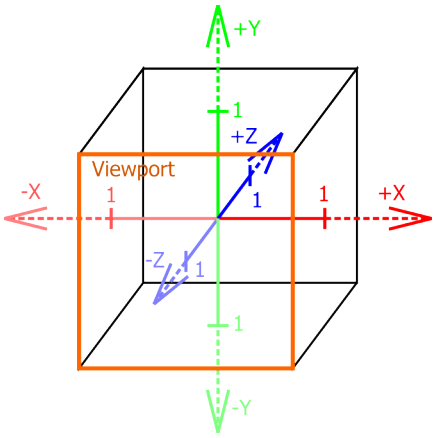
Vertex shader se provádí se pro každý vrchol vykreslované geometrie a jediným jeho povinným výstupem jsou souřadnice vrcholu v tzv. normalizovaném souřadném systému zařízení (normalized device coordinate system) viz dále.

K dispozici má tři druhy vstupů. První z nich představují jeden nebo více atributů (attributes). Tyto jsou většinou brány z Vertex Buffer Objektů (VBO). VBO představují jakési pole atributů, která jsou uložena v paměti grafické karty. V nich jsou většinou netransformované souřadnice a jiné (texturové souřadnice, normály) atributy vrcholů vykreslované geometrie. Atributy však mohou být nastaveny i na pevnou hodnotu. Dalším možným vstupem do vertex shaderu jsou proměnné typu uniform (uniforms), jež je možné nastavit příslušnými funkcemi, a poté se chovají jako konstanty v rámci celého vykreslování. K nim lze přistupovat jak z vertex shaderu, tak z fragment shaderu. Posledním možným vstupem jsou texturovací jednotky (texture image units), tyto však nemusí být z vertex shaderu dostupné ve všech WebGL implementacích.

Výsledné souřadnice jsou zapsány do proměnné gl_Position, jež je 4-složkový vektor. Tento se zpravidla tvoří transformací souřadnic načtených z VBO skrz atributy. Normalizované souřadnice zařízení vzniknou z proměnné gl_Position podělením prvních 3 složek složkou čtvrtou. Toto umožňuje vytvoření perspektivy.

Normalizovaný souřadný systém zařízení je levotočivý souřadný systém, přičemž vykreslovaná scéna tvoří objem pomyslné krychle v tomto souřadném systému. Tato krychle vypadá tak, že počátek souřadného systému leží uprostřed této krychle a její stěny protínají jednotlivé osy ve vzdálenosti 1 od tohoto počátku. Viewport, do kterého se scéna promítá, představuje plochu krychle ležící ve směru záporné souřadnice Z od počátku (viz Obr. 3). Části geometrie, zasahující mimo tuto krychli, se nevykreslí.

Dalším nepovinným výstupem jsou proměnné typu varying, které umožňují předávat další parametry vrcholů.



Obr. 3 Normalizovaný souřadný systém zařízení a viewport

2.4.3 Vytvoření primitiv

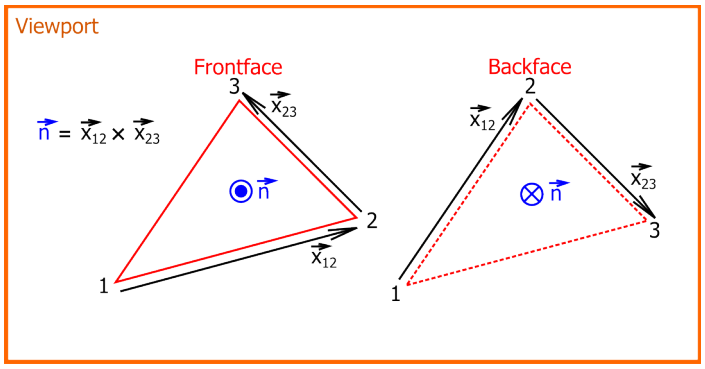
Každá geometrie je tvořena soustavou primitiv. Tato primitiva jsou ve WebGL buď trojúhelníky nebo úsečky. Tyto se tvoří z po sobě jdoucích vrcholů pomocí pravidla, které se nastaví před vykreslováním (např. vždy trojice vrcholů tvoří trojúhelník nebo první 3 vrcholy vytvoří trojúhelník a pak každý další vrchol s pomocí 2 předchozích vytvoří trojúhelník atd.).

Pořadí vrcholů při tvorbě primitiv je dáno pořadím, ve kterém jsou jejich atributy brány z příslušných VBO (to protože atributy brané z VBO představují jediné vstupy vertex shaderu, které nejsou v rámci jednoho vykreslování konstantní). Ty mohou být brány buď sekvenčně, kdy pořadí vrcholů odpovídá pořadí jejich atributů uvnitř VBO, nebo indexovány dalším VBO. Indexace dalším VBO umožňuje sdílení vrcholů více primitivy, bez nutnosti duplikovat jejich atributy uvnitř VBO.

2.4.4 Rasterizace a interpolace proměnných varying

Po vytvoření primitiv je provedena rasterizace. To je proces prováděný na úrovni primitiv, při kterém se zjistí, které pixely budou ovlivněny daným primitivem a pro každý takový pixel je vykonán fragment shader. Hodnota každé proměnné typu varying pro fragment shader je vytvořena interpolací hodnot vrcholů, dodaných vertex shaderem, příslušících danému primitivu. Interpolovat lze, jak skaláry, tak vektory. V rámci této operace se také „ořeže“ veškerá geometrie (případně její části), ležící mimo onu pomyslnou krychli popisovanou v sekci vertex shader.

Dále lze volitelně zapnout tzv. face culling. To je optimalizační technika kdy se trojúhelníky mohou vyřazovat z dalšího zpracování v závislosti tom, jestli jejich vrcholy promítnuté do viewportu jdou v posloupnosti (viz pořadí vrcholů výše) po směru (backface) nebo protisměru (frontface) hodinových ručiček (grafická karta toto určí pomocí souřadnice Z normály, vytvořené ze souřadnic vrcholů s využitím vektorového součinu viz Obr. 4). Vychází se zde předpokladu, že vykreslovaná geometrie často ohraničuje objem v prostoru a trojúhelníky, které ji tvoří mají tedy „vnitřní“ stranu, která není nikdy viditelná. Face culling tak umožňuje ušetřit velké množství výkonu grafické karty vyřazením z dalšího zpracování trojúhelníků, u nichž je do viewportu promítána jejich „vnitřní“ strana. Ty by byly jinak vyřazeny až na bázi pixelů depth testem (viz dále) nebo „překresleny“ geometrií ležící blíže k virtuálnímu pozorovateli.



Obr. 4 Face culling (pro názornost použit pravotočivý souřadný systém)

2.4.5 Fragment shader

Fragment shader se provádí pro každý pixel vykreslované geometrie a jeho cílem je stanovit výslednou barvu pixelu. K tomu má k dispozici více vstupů.

Prvním vstupem fragment shaderu jsou proměnné typu varying, jejichž hodnota se získá postupem popsaným výše. Tyto nejčastěji představují souřadnice v textuře, normály apod. Dalším vstupem jsou opět proměnné typu uniform, jak bylo zmíněno u vertex shaderu. Posledním vstupem jsou texturovací jednotky, skrze které je možno přistupovat k datům textur. (Toto je realizováno prostřednictvím speciálního uniformu samplerXD, kde X je počet dimenzí textury.)

Jediným výstupem fragment shaderu je proměnná gl_FragColor, která obsahuje výslednou barvu pixelu. (Jde o 4-složkový vektor, který tvoří barevné komponenty RGB a průhlednost A - alfa.) Tato barva se poté zapíše do framebufferu. (Pokud je zapnut alpha blending, může se tato barva „míchat“ s podkladovou, přičemž je využito výše zmíněné alfy.)

Zápis do framebufferu je podmíněný v případě, že je zapnut depth test, tedy kontrola hloubky. Kontrola hloubky je realizována tak, že pro každý pixel se počítá hloubka (souřadnice jež se zvyšuje s vzdáleností od virtuálního pozorovatele), ve které leží a ta je porovnána s aktuální hodnotou uloženou v depth bufferu (ten je součástí framebufferu). K zápisu dojde, jen pokud hodnota splňuje nastavené porovnávací pravidlo (zpravidla pokud má menší hodnotu). Toto umožňuje správně vyřešit viditelnost geometrie v závislosti na vzdálenosti od virtuálního pozorovatele v případě, kdy dochází k překrývání primitiv (většinou trojúhelníků).

Zápisu do framebufferu lze zabránit také zavoláním funkce discard.

2.4.6 Výhody

Mezi výhody WebGL patří:

- nativní podpora prohlížečů,
- poskytuje nízkoúrovňový přístup ke grafické kartě a umožňuje její programovatelnost,
- dobrá podpora prohlížečů (včetně verzí pro mobilní zařízení) s výjimkou Internet Exploreru (zde je nutný plugin),
- snadná adaptace z OpenGL.

2.4.7 Nevýhody

Nevýhody WebGL jsou:

- nutná hlubší znalost fungování grafických karet a jejich programování,
- WebGL neposkytuje žádné vysokoúrovňové funkce pro kreslení vektorové grafiky nebo textu, k čemuž je nutno využít textur.

2.5 Adobe Flash

Je technologie určená především pro vytváření aplikací pro web. Původním autorem Flashe je společnost Macromedia, nyní vlastněná společností Adobe. Flash dosáhl značného rozšíření a je tak používán k tvorbě reklamních banerů, internetových přehrávačů i webových her. Flashi se povedlo vyplnit mezeru na trhu v oblasti vektorové grafiky a multimediálních aplikací a stal se tak na dlouhou dobu de facto standardem v této oblasti. Přes své rozšíření byl Flash dlouho dobu uzavřeným standardem a k vytvoření aplikace bylo nutno použít software Adobe. V současné době jsou dostupné některé specifikace uvolněné v rámci projektu The Open Screen Project, započatého firmou Adobe v roce 2008. Jako skriptovací jazyk je použit ActionScript, který je určitým rozšířením EcmaScriptu. K dispozici má široké API umožňující jak tvorbu vektorové a 3d grafiky, tak přehrávání audio a video souborů.

2.5.1 Výhody

K hlavním výhodám Flashe patří:

- malá velikost aplikací – využívá komprese,
- zdrojové kódy nejsou snadno získatelné,
- velké rozšíření – většina prohlížečů má instalován potřebný plugin,
- široké API.

2.5.2 Nevýhody

Nevýhody Flashe jsou:

- prohlížeči není podporován nativně – je vyžadován plugin firmy Adobe,
- pouze omezená podpora nekomerčním softwarem,
- postupně vytlačován jinými standardy jako HTML5.

2.6 Silverlight

Silverlight je technologie určená pro vývoj webových multimediálních aplikací od firmy Microsoft. První verze vyšla v roce 2007 jako konkurenční technologie v té době dominujícího Flashe. Tato technologie je založena na zredukované verzi Common Language Runtime platformy .NET. Jako programovací je tedy použit C#. Silverlight poskytuje API umožňující mimo jiné přehrávání multimédií, vytváření 2D a 3D grafiky a další. Nikdy však nikdy nedosáhl velkého rozšíření. Stránky dříve používající Silverlight nyní často přechází na HTML5 [10].

2.6.1 Výhody

Výhodou technologie Silverlight je:

- široké API založené na platformě .NET.

2.6.2 Nevýhody

Mezi nevýhody Silvelightu patří:

- pro běh je nutno mít instalován zásuvný modul,
- malé rozšíření [10],
- postupně je vytlačován HTML5 standardem.

2.7 Java

Java je programovací jazyk vyvinutý firmou Sun Microsystems, která je nyní vlastněna společností Oracle Corporation. V Javě lze vyvíjet pro různé platformy kterými jsou: Java ME (mobilní zařízení), Java SE (stolní a serverové počítače) a Java EE (aplikace pro aplikační severy). Na rozdíl od většiny ostatních zmíněných technologií se tedy oblast jejího použití neomezuje na web.

2.7.1 Použitelnost Javy pro webové hry

Z hlediska vývoje webových her je zajímavá platforma Java SE. Aplikace napsaná v Javě SE spouštěná z webu může mít dvě formy a to applet a tzv. Java Web Start (JWS). Zatímco ve formě appletu představuje Java aplikace element na webové stránce, tak ve formě JWS běží jako nezávislá aplikace (může vytvářet vlastní okna atd.).

Java SE má k dispozici široké API umožňující nejrůznější použití. Mezi možnosti které toto API nabízí patří, jak tvorba klasického GUI, tak nízkoúrovňová grafika prostřednictvím třídy Graphics2D. Třída Graphics2D umožňuje široké možnosti na poli vektorové grafiky a to včetně křivek a pokročilých funkcí jako dashing, filling s využitím pravidla (even-odd nebo non-zero), composite umožňující ovlivnit způsob kombinace vykreslované a podkladové barvy a antialiasingu. Java SE API se však neomezuje pouze na grafiku a lze zde najít třídy pro přístup k souborovému systému, síťovou komunikaci, šifrování, práci s XML, podporu rastrových obrazových formátů, zvuku a další. Pro některé potenciálně nebezpečné operace (např. přístup k souborovému sytému) je však při spuštění z prohlížeče vyžadováno svolení uživatele.

Dalším rozdílem oproti ostatním technologiím, které jsou založeny většinou na JavaScriptu (EcmaScriptu), je to, že nejde o skriptovací jazyk. Jde o staticky typový programovací jazyk, který je však překládán Java bytecodu (instrukční sady javového virtuálního stroje). Ten se do kódu procesoru překládá až za běhu (využívá JIT), což umožňuje jeho přenositelnost mezi různými platformami. Java je proto vhodná pro komplexnější aplikace vyžadující vyšší výkon. Protože Java je mimo jiné určena k serverovému nasazení, má velmi dobrou podporu vícevláknových aplikací a to jak na úrovni jazyka, tak skrze další třídy (balíček java.util.Concurent). To umožňuje efektivně využít všechna jádra vícejádrových procesorů.

2.7.2 Výhody

Hlavní výhody Javy jsou:

- široké API umožňující tvořit i složité programy,
- adaptovatelnost existujících programů pro běh z webu,
- výkon srovnatelný s programovacími jazyky překládanými přímo do kódu procesoru,
- zdrojový kód není pro uživatelem přímo získatelný,
- malá velikost aplikací (programů) daná použitím binárního (netextového) formátu spustitelných souborů a komprese.

2.7.3 Nevýhody

Mezi nevýhody Javy patří:

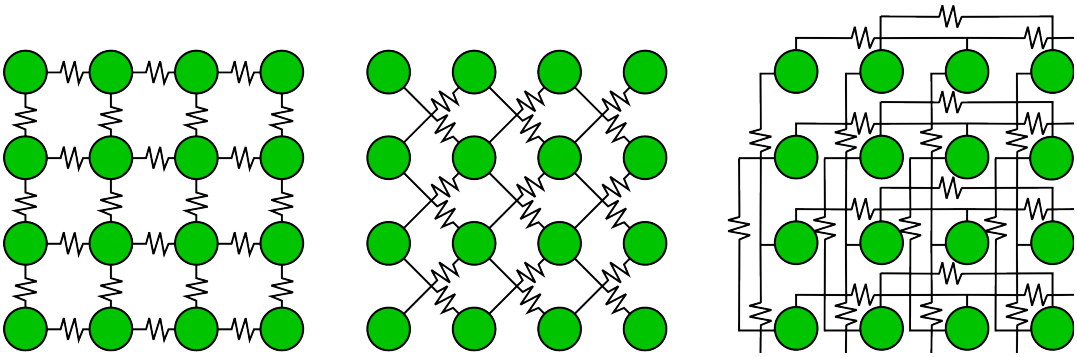
- pro běh z prohlížeče je nutný plugin,
- oproti ostatním technologiím vyšší nároky na diskový prostor dané nutností instalace celého běhového prostředí Javy.

3 APLIKACE VLAJKA VE VĚTRU

Tato aplikace byla vytvořena jako demonstrace možností WebGL. To zde není použito pouze k vykreslování, ale trochu netradičně také k výpočtům. Existují však i jiné webové aplikace používající WebGL k výpočtům [11] [12].

3.1 Rozbor

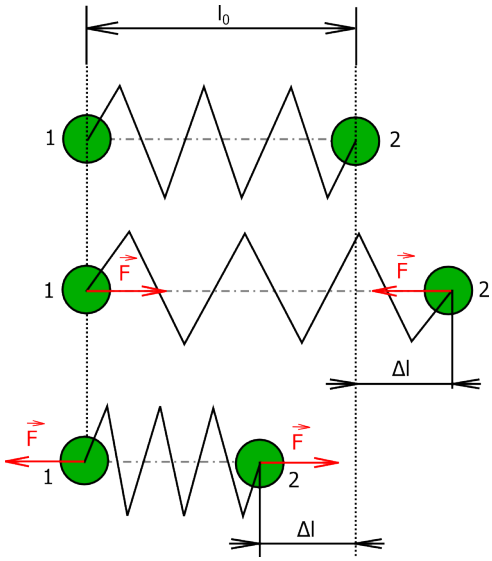
Aplikace provádí numerickou simulaci vlajky ve větru v reálném čase pro vizuální účely. Na modelování vlajky je použita soustava hmotných bodů spojených pružinami a tlumiči. Jde o běžný způsob simulování tkanin (cloth simulation). Soustavu pružin (a tlumičů) lze rozdělit do tří skupin. V anglické literatuře jsou tyto tři skupiny často označovány jako structural, shear a bend [13] [17]. Cílem jednotlivých skupin je zajistit určitou tuhost jak ve vodorovném a diagonálním směru, tak i proti ohybu. Simulace se provádí po krátkých časových krocích, kdy se výpočte zrychlení jednotlivých hmotných bodů a následně se numericky integruje, pro výpočet nové rychlosti a polohy.



Obr. 5 Skupiny pružin (tlumičů) zleva: structural, shear a bend

3.2 Modelování pružin

Pružina představuje zařízení, které působí silou při vychýlení z rovnovážné polohy. Síla je přímo úměrná výchylce, přičemž směr síly je lze vidět na Obr. 6 . Ve skalárním tvaru je tedy síla dána rovnicí (3.1).



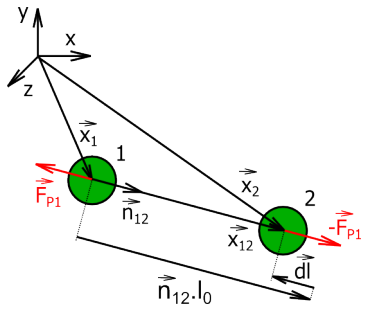
Obr. 6 Chování pružiny

Síla vyvíjená pružinou ve skalárním tvaru

$$F = -k \cdot x \tag{3.1}$$

kde x je výchylka z rovnovážné polohy a k tuhost pružiny

Protože v simulaci je potřeba spočítat vektor zrychlení, byl celý problém převeden do vektorového tvaru a sestaveny vektorové rovnice. Vyobrazení vektorových veličin je vidět na Obr. 7 . Následují vektorové rovnice.



Obr. 7 Pružina se zobrazením vektorových veličin

Polohový vektor bodu 2 vzhledem k bodu 1

$$\vec{x}_{12} = \vec{x}_2 - \vec{x}_1 \tag{3.2}$$

kde x_1 je polohový vektor bodu 1 a x_2 polohový vektor bodu 2

Jednotkový vektor směru z bodu 1 do bodu 2

$$\vec{n}_{12} = \frac{\vec{x}_{12}}{|\vec{x}_{12}|} \tag{3.3}$$

Vektor vychýlení z rovnovážné polohy

$$\vec{dl} = \vec{x}_{12} - \vec{n}_{12} \cdot l_0 \tag{3.4}$$

kde l_0 je délka pružiny v rovnovážné poloze

Síla od pružiny působící na bod 1

$$\vec{F}_{P1} = k_P \cdot \vec{dl} \tag{3.5}$$

kde k_P je tuhost pružiny

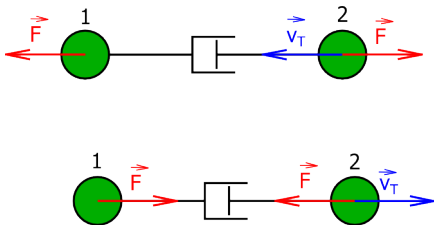
Zrychlení od pružiny působící na bod 1

$$\vec{a}_{P1} = \frac{\vec{F}_{P1}}{m_1} \tag{3.6}$$

kde m_1 je hmotnost bodu 1

3.3 Modelování tlumičů

Tlumič je zařízení, které klade odpor při změně délky. Síla vyvíjená tlumičem je přímo úměrná rychlosti změny délky tlumiče a působí ve směru proti tomuto pohybu (viz. Obr. 8). Hmotný bod 1 je zde uvažován stojící a rychlost $\vec{v}_T$ tedy představuje rychlost změny délky tlumiče. Síla ve vyvíjená tlumičem ve skalárním tvaru je dána rovnicí (3.7).



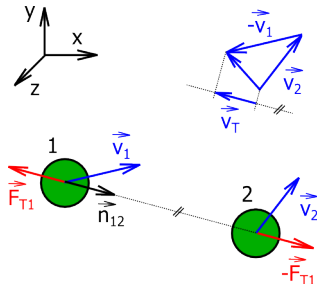
Obr. 8 Chování tlumiče

Síla vyvíjená tlumičem ve skalárním tvaru

$$F = -k_T . v_T \tag{3.7}$$

kde k_T je konstanta tlumení a v_T je rychlost změny délky tlumiče

Protože v simulaci je opět potřeba vektor zrychlení, byl problém převeden do vektorového tvaru. Následují vektorové rovnice. Vyobrazení vektorových veličin je vidět na Obr. 9 .



Obr. 9 Tlumič se zobrazením vektorových veličin

Rychlost změny délky tlumiče

$$\vec{v}_T = (\vec{v}_2 - \vec{v}_1) \cdot \vec{n}_{12} \cdot \vec{n}_{12} \tag{3.8}$$

kde $\vec{v}_1$ je rychlost bodu 1 a $\vec{v}_2$ je rychlost bodu 2 a $\vec{n}_{12}$ jednotkový vektor směru z bodu 1 do bodu 2 (viz výše)

Síla od tlumiče působící na bod 1

$$\vec{F}_{T1} = k_T \cdot \vec{v}_T \tag{3.9}$$

kde k_T je konstanta tlumení

Zrychlení od tlumiče působící na bod 1

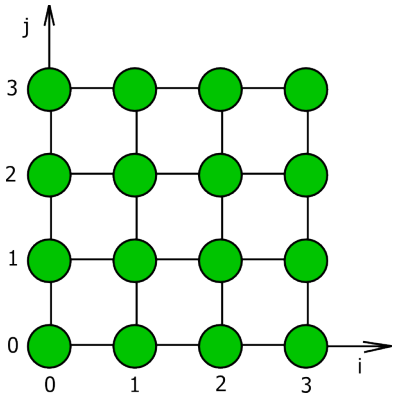
$$\vec{a}_{T1} = \frac{\vec{F}_{T1}}{m_1} \tag{3.10}$$

kde m_1 je hmotnost bodu 1

3.4 Mapování dat ve WebGL

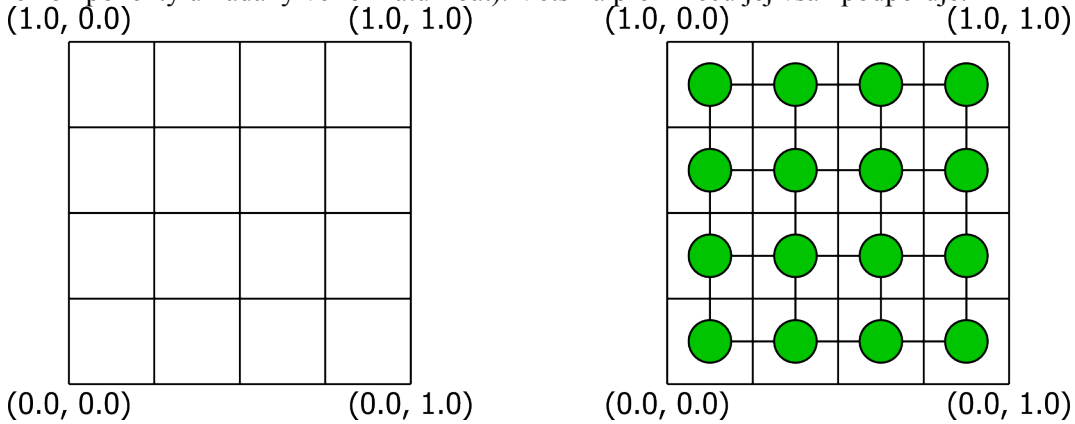
Nejprve bylo nutné stanovit mapování dat simulace do prostředí grafické karty. Každému hmotnému bodu tvořícímu simulaci přísluší několik veličin, tj. poloha, rychlost, zrychlení a normála (pro výpočet osvětlení).

Jednotlivé veličiny hmotných bodů si lze představit jako skupinu diskrétních 2D vektorových polí. Myšlené souřadnice v těchto polích budou dále označovány jako i a j. (Tyto v samotné aplikaci používány nejsou.) Protože pole jsou diskrétní nabývají souřadnice i a j pouze hodnot přirozených čísel. Sousední hmotné body v poli jsou sousední v síti. Pro vlajku 4 x 4 hmotné body by tedy pole vypadalo, jak je vidět na Obr. 10 . Zelené body představují hodnoty (tvořené vektory) dané veličiny.



Obr. 10 Diskrétní vektorové pole jednotlivých veličin

Takováto pole lze pak snadno mapovat na grafickou kartu v podobě textur. Textury však používají odlišný souřadný systém. Tento souřadný systém má souřadnice (0, 0) v levém dolním rohu textury a (1, 1) v pravém horním a používá proměnné typu float (tedy čísla s plovoucí řádovou čárkou). Oba souřadné systémy však mezi sebou mají jednoduchou lineární závislost. Protože je mapováno diskrétní pole filtrování těchto textur je nastaveno na tzv. nearest neighbour, tedy pomocí souřadnice je získána barva nejbližšího pixelu (neprovádí se interpolace). Mapování lze vidět na Obr. 11 . Čtvercové oblasti představují jednotlivé pixely. Komponenty vektorů jsou mapovány na barevné složky těchto pixelů. K dosažení potřebné přesnosti je dále třeba, aby WebGL podporovalo rozšíření (extension) OES_texture_float (to rozšiřuje WebGL přidáním formátu textury ve kterém jsou barevné komponenty ukládány ve formátu float). Většina prohlížečů jej však podporuje.



Obr. 11 vlevo souřadný systém textury, vpravo diskrétní pole mapované do textury

3.5 Výpočet ve WebGL

Výpočty jsou realizovány renderováním do textury a to následujícím způsobem. Nastaví se renderování do FBO, ke kterému se připojí textura, ve které budou výsledná data výpočtu. Na příslušné texturovací jednotky se připojí textury, které budou obsahovat data potřebná k výpočtu. Všechna nepotřebná funkcionalita WebGL je po čas výpočtů vypnuta (např. depth test).

Viewport se nastaví přes celou texturu (k viewportu se vážou souřadnice ukládané do proměnné gl_Position, proto je případné omezení výpočtů na určitou oblast provedeno sccissor testem, který umožňuje dále omezit renderování na obdélníkovou oblast v rámci viewportu). Renderovány jsou pouze 2 trojúhelníky, které dohromady vyplňují celou texturu.

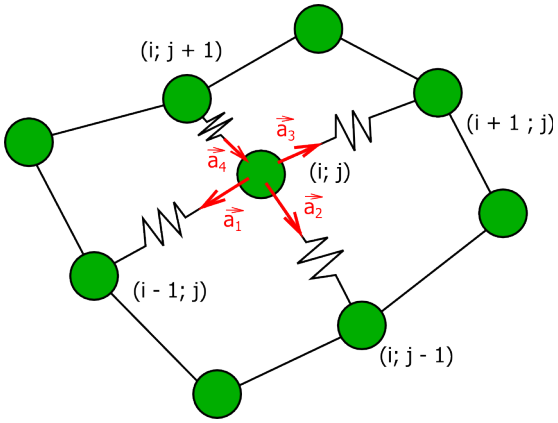
K renderování je použit velmi jednoduchý vertex shader, který prostřednictvím proměnné typu varying zprostředkuje odpovídající texturovou souřadnici.

Samotný výpočet vykonává fragment shader. V tomto případě se tedy píše místo pro pixel, pro hmotný bod. Zdrojová data získává z textur (představující zdrojová vektorová pole) prostřednictvím texturovacích jednotek. Pro konstanty v rámci výpočtu jsou využity proměnné typu uniform. Výsledná hodnota (vektor) je přiřazena do proměnné gl_FragColor (nebo alternativně gl_FragData[0]), která je poté zapsána do textury připojené k FBO. Tato textura představuje cílové vektorové pole.

3.6 Výpočet zrychlení fragment shadery

Pro výpočet zrychlení hmotných bodů je využito principu superpozice. To umožňuje rozdělit výpočet do více fragment shaderů (tedy i programů => k výpočtům je provedeno vícero vykreslování). První fragment shader pouze nastaví všem hmotným bodům stejné zrychlení (představující gravitační zrychlení), další poté přičítají zrychlení od jednotlivých skupin pružin a tlumičů. K tomu je využito rovnic odvozených výše. Aby se omezilo množství výpočtů, jsou všechny konstanty, kde je to možné, sloučeny do jediné (např. podíl tuhosti pružiny a hmotnosti bodu je nahrazen jedinou konstantou).

Následuje příklad fragment shaderu pro výpočet zrychlení od pružin a tlumičů typu structural (poté taky bend se zvětšenou proměnou spacing). Vyobrazení lze vidět na Obr. 12 .



Obr. 12 Zrychlení působící na hmotný bod od pružin (a tlumičů) typu shear

```
#version 100  // verze jazyka GLSL ve které je shader psán
precision highp float;      // výpočet s vyšší přesností
varying vec2 texCoord;  // proměnná předávající texturovou souřadnici hm. bodu (i; j) viz. Obr. 12
uniform vec2 spacing;    // vzdálenosti mezi hmotnými body v texturové souřadnici
uniform float l0;    // neutrální délka pružin
uniform float kp;   // konstanta pružin
uniform float kt;    // konstanta tlumičů
uniform sampler2D positionField;   // sampler přes který se přistupuje k vektorovému poli poloh
uniform sampler2D accelerationField;   // vektorové pole zrychlení
uniform sampler2D velocityField;        // vektorové pole rychlosti
vec3 getAcceleration(vec3 x1, vec3 v1, vec2 coor);  // deklarace funkce na výpočet zrychlení
void main(){      // funkce vykonaná pro každý pixel (hmotný bod)
 vec3 a1 = texture2D(accelerationField, texCoord).rgb;   // načtení zrychlení z vektorového pole
 vec3 x1 = texture2D(positionField, texCoord).rgb;     // načtení polohy z vektorového pole
 vec3 v1 = texture2D(velocityField, texCoord).rgb;    // načtení rychlosti z vektorového pole
 vec2 spacingX = vec2(spacing.x, 0.0); // „vodorovná“ (ve směru i) vzdálenost mezi hm. Body
 // jako  vektor
 vec2 spacingY = vec2(0.0, spacing.y); // „svislá“ ( ve směru j) vzdálenost mezi hm. body jako vektor
 //
 vec2 coor = texCoord + spacingX;  // texturová souřadnice hmotného bodu „napravo“ (i +1; j)
 if(coor.x < 1.0){   // zrychlení bude počítáno jen pokud již sám hm. bod není v tomto směru krajní
  a1 += getAcceleration(x1, v1, coor);    // přičtení všech zrychlení od sousedního bodu
 }
 coor = texCoord – spacingX;    // texturová souřadnice hmotného bodu „nalevo“ (i - 1; j)
 if(coor.x > 0.0){
  a1 += getAcceleration(x1, v1, coor);
 }
```

```
coor = texCoord + spacingY;  // (i; j +1)
 if(coor.y < 1.0){
  a1 += getAcceleration(x1, v1, coor);
 }
 coor = texCoord – spacingY;  // (i; j -1)
 if(coor.y > 0.0){
  a1 += getAcceleration(x1, v1, coor);
 }
 gl_FragData[0] = vec4(a1, 0.0);    // zápis výsledného zrychlení do FBO
}

vec3 getAcceleration(vec3 x1, vec3 v1, vec2 coor){   // funkce na výpočet zrychlení od bodu 2
 // bod 1 – bod pro který se provádí výpočet, bod 2 – sousední bod jehož vliv je přičítán
 vec3 x12 = x1 - texture2D(positionField, coor).rgb;  // vektor z bodu 1 do bodu 2
 vec3 n12 = normalize(x12);                           // jednotkový vektor směru z bodu 1 do bodu 2
 vec3 vt = vec3(dot(texture2D(velocityField, coor).rgb - v1, n12)) * n12; // rych. změny délky tlumiče
 vec3 dl = n12 * vec3(l0) - x12;              // vektor vychýlení pružiny z rovnovážné polohy
 return vec3(kt) * vt + vec3(kp) * dl;       // zrychlení vyvolané tlumičem a pružinou dohromady
}
```

3.7 Zrychlení od větru

Dalším zrychlením je zrychlení od větru, působícího na vlajku. Protože výpočet proudění vzduchu kolem vlajky by byl velmi složitý a v reálném čase nereálný, byl tento výpočet značně zjednodušen. Ale jelikož jde o simulaci pro vizuální účely, není to důvod k znepokojení.

Při výpočtu větru je vlajka myšleně rozdělena na plošky vždy mezi čtyřmi hmotnými body. Každá taková ploška je tvořena dvěma trojúhelníky. Pro tyto je pak počítána síla (z ní pak zrychlení) od větru. Výpočet vychází ze vztahů pro odpor vzduchu (3.11) a vztlak (3.12). Koeficienty (a jejich závislost na náklonu plošky) jsou neznámé, ale dá se předpokládat, že při snižování hodnoty jednoho koeficientu vlivem náklonu plošky bude druhý stoupat a naopak. Z tohoto vychází další zjednodušení, které předpokládá výslednici sil ve směru normály plošky za použití jediného koeficientu C_x. Zjednodušený výpočet síly působící na trojúhelník je popsán rovnicí (3.15). Při výpočtu je dále možno sloučit všechny konstanty. Zrychlení celé plošky je poté spočteno vektorovým součtem sil působících na oba trojúhelníky, poděleným hmotností plošky.

Síla vyvolaná odporem vzduchu

$$F = \frac{1}{2} \cdot \rho \cdot C_D \cdot v^2 \cdot A \qquad (3.11)$$

kde ρ je hustota vzduchu, C_D je součinitel odporu vzduchu, v je rychlost proudění vzduchu a A je plocha tělesa v průmětu kolmém na směr proudění

Vztlaková síla

$$F = \frac{1}{2} \cdot \rho \cdot C_L \cdot v^2 \cdot A \qquad (3.12)$$

kde C_L je součinitel vztlaku (ostatní veličiny viz výše)

Vektor kolmý k trojúhelníku trojúhelníku o délce odpovídající jeho ploše

$$\vec{A} = \frac{1}{2} \cdot \vec{x_{21}} \times \vec{x_{31}} \qquad (3.13)$$

kde $\vec{x_{21}}$ je polohový vektor bodu 2 vůči bodu 1 a $\vec{x_{31}}$ je polohový vektor bodu 3 vůči bodu 1 (trojúhelník je tvořen body 1 2 a 3)

Normála trojúhelníku

$$\vec{n}_V = \frac{\vec{A}}{|\vec{A}|} \tag{3.14}$$

Zjednodušený vstah pro výpočet vektoru síly působící na trojúhelník

$$\vec{F} = \frac{1}{2} \cdot \rho \cdot C \cdot v^2 \cdot \vec{A} \cdot \vec{n}_w \cdot \vec{n} \tag{3.15}$$

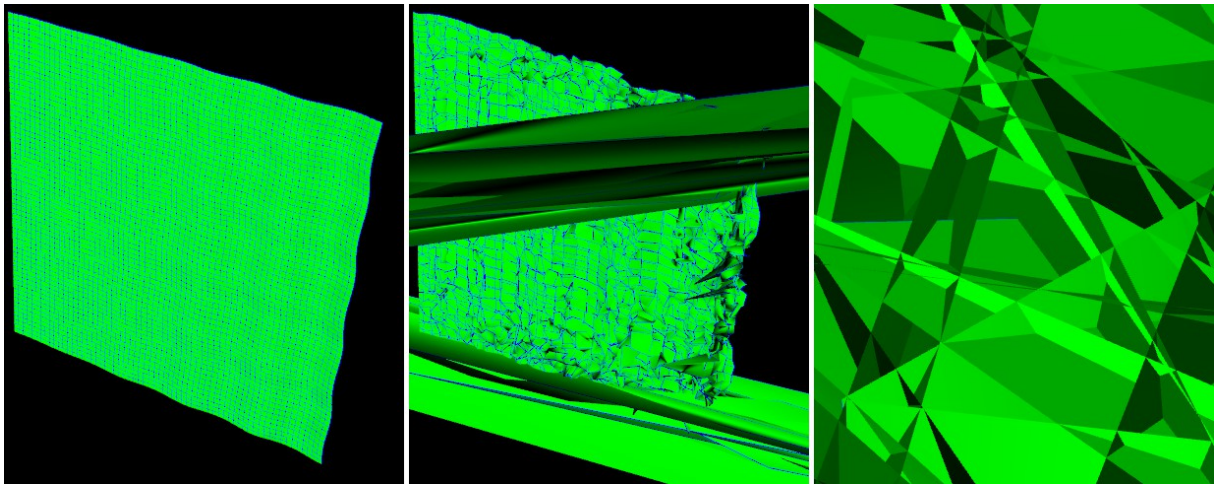
kde C je součinitel síly působící ve směru normály plošky a $\vec{n}_w$ je jednotkový vektor ve směru větru (ostatní veličiny viz výše)

Protože výpočet je však tentokrát provázen na bázi plošek a ne hmotných bodů, je nutné toto zrychlení rozpočítat mezi okolní hmotné body. Proto je pro tento výpočet vytvořeno další pole (v podobě textury samozřejmě), do kterého jsou vypočítány hodnoty zrychlení pro jednotlivé plošky. Ty jsou potom rozpočítány mezi hmotné body. Toto je provedeno s využitím možnosti lineární interpolace pomocí texturovací jednotky, kdy texturová souřadnice odkazuje mezi 4 pixely (představující vektory zrychlení okolních plošek hmotného bodu).

3.8 Numerická integrace

Pro výpočet rychlostí a poloh hmotných bodů je dále potřeba provést dvojnásobnou numerickou integraci. Protože vlajka je tvořena mnoha oscilátory byly potřeba řešit problémy s numerickou stabilitou. Pokud se totiž simulace dostane do numericky nestabilního stavu, znamená to kupení numerický chyb, což má za následek znehodnocení celé simulace (viz. Obr. 13).

Numerickou stabilitu lze ovlivnit více způsoby. Prvním je konfigurace pružin a tlumičů. Obzvlášť s „tvrdšími“ pružinami má simulace sklony k numerické nestabilitě. Zde jsou však možnosti značně omezené, protože příliš pružná vlajka nepůsobí reálně. Dalšími možnostmi jsou volba vhodné metody pro numerickou integraci a zmenšení časového kroku (to ale zvyšuje výpočetní náročnost). Celkem byly testovány 3 různé metody numerické integrace a to Explicitní Eulerova metoda, NSV a RK4.



Obr. 13 Průběh simulace při numerické nestabilitě (snímky jsou ve stejném měřítku)

3.8.1 Explicitní Eulerova metoda

Toto je nejjednodušší metoda, která novou rychlost a polohu počítá podle vzorců níže. Tato metoda se však, podle předpokladu, ukázala být nevhodná a značně nestabilní pro tuto úlohu.

$$y_{n+1} = y_n + h \cdot f(t_n, y_n) \tag{3.16}$$

$$v_{n+1} = v_n + a \cdot \Delta t \tag{3.17}$$

$$x_{n+1} = x_n + v_n \cdot \Delta t \tag{3.18}$$

3.8.2 NSV

Rozdíl NSV oproti Explicitní Eulerově metodě spočívá ve využití již nově spočítané rychlosti při další integraci pro výpočet polohy. Tato metoda je určena pro podtlumené oscilátory (především pak netlumené) [14], což je i případ této simulace, kde slibuje vyšší numerickou stabilitu než explicitní Eulerova metoda. Toto se také potvrdilo.

$$v_{n+1} = v_n + a \cdot \Delta t \tag{3.19}$$

$$x_{n+1} = x_n + v_{n+1} \cdot \Delta t \tag{3.20}$$

3.8.3 RK4

RK4 je metoda vyššího řádu, konkrétně 4. [16] jak napovídá název. Jako taková pak využívá celkem 4 výpočtů zrychlení, z kterých pak počítá výslednou hodnotu rychlosti (obdobně pak s rychlostí a polohou). To vede k lepší přesnosti a stabilitě metody i při větším časovém kroku. Náročnost výpočtu jednoho časového kroku je však značně zvýšena.

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4) \tag{3.21}$$

$$k_1 = f(t_n, y_n)$$
$$k_2 = f(t_n + 0{,}5 \cdot h,\ y_n + 0{,}5 \cdot h \cdot k_1)$$
$$k_3 = f(t_n + 0{,}5 \cdot h,\ y_n + 0{,}5 \cdot h \cdot k_2)$$
$$k_4 = f(t_n + h,\ y_n + h \cdot k_3)$$

3.9 Výpočet osvětlení

Při výpočtu osvětlení bylo použito modelu diffuse lighting. Ten modeluje osvětlení matných povrchů. Pro každý pixel takto renderované geometrie se světlost spočítá jako skalární součin vektoru přicházejícího světla a normálového vektoru povrchu v místě pixelu. Výsledná barva pixelu je poté dána součinem světlosti a barvy povrchu v daném místě.

V případě této simulace jsou normály počítány v místech hmotných bodů. Při výpočtu osvětlení jednotlivých pixelů se poté interpolují (prostřednictvím proměnné typu varying) normály okolních bodů. Výsledná hodnota se poté normalizuje (tj. převede na jednotkový vektor). V aplikaci je použito paralelní osvětlení. Pro větší přehlednost je použita absolutní hodnota světlosti (tzn. paralelní osvětlení je z obou stran).

3.9.1 Výpočet normál

Výpočet normály vychází z předpokladu, že u spojité funkce je normála v každém místě kolmá na její tečnu (popř. tečny). Protože povrch vlajky je spojitý (a netvoří ostré hrany), je uvažováno, že vektorové pole poloh tvoří spojitou funkci. Tečny takovéto funkce lze stanovit numericky. Numerická derivace se často stanovuje z dvou okolních bodů, ta je pro funkce jedné proměnné daná rovnicí (3.22). Protože v tomto případě tohoto pole jde o funkci 2 proměnných (virtuálních souřadnic i a j), je použit tvar pro 2 proměnné (3.23). Protože jmenovatel ovlivňuje pouze délku vektoru nikoliv jeho směr je možné jej vypustit (tj. 2h uvažovat jako 1).

Pro každý bod jsou vypočítány derivace ve směru virtuálních os i a j. Normála je poté spočítána jako vektorový součin vektorů obou tečen, který je normalizován. Graficky je proces znázorněn na Obr. 14 .

Numerická derivace funkce 1 proměnné (vzorec pro centrální diferenci) [18]

$$f'(x) = \frac{f(x+h) - f(x-h)}{2\mathrm{h}} \tag{3.22}$$

Numerická derivace funkce 2 proměnných

$$f'(\vec{x};\vec{r}) = \frac{f(\vec{x}+h\cdot\vec{r}) - f(\vec{x}-h\cdot\vec{r})}{2h} \qquad (3.23)$$

kde $\vec{x}$ je bod ve kterém se derivace počítá, $\vec{r}$ představuje směr ve kterém se derivuje a h je délka kroku

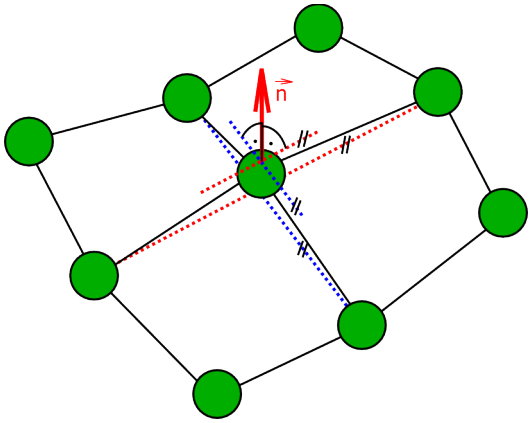
Výpočet normálového vektoru povrchu vlajky v bodě o virtuálních souřadnicích (i; j)

$$\vec{n}_1 = (\vec{x}(i+1;j) - \vec{x}(i-1;j)) \times (\vec{x}(i;j+1) - \vec{x}(i;j-1)) \qquad (3.24)$$

kde $\vec{x}$ představuje vektorové pole pozic hmotných bodů

Vytvoření normálového vektoru z vektoru $\vec{n}_1$ jeho normalizací

$$\vec{n} = \frac{\vec{n}_1}{|\vec{n}_1|} \qquad (3.25)$$



Obr. 14 Normála povrchu v místě hmotného bodu

4 APLIKACE SIMULUJÍCÍ JEDNODUCHOU 2D FYZIKU

Aplikace je postavená na vektorém grafickém formátu SVG doplněném skripty v javascriptu. Kód této aplikace tvoří především 2D fyzikální engine. Jeho cílem je detekovat kolize mezi zobrazovanými objekty a následně je řešit s použitím jednoduchého fyzikálního modelu. K vytvoření scény byl použit vektorový editor Inkscape. Veškerá data potřebná pro fyzikální výpočty jsou programově načítána ze SVG souboru s pomocí DOM API.

4.1 Fyzikální model

Aplikace rozlišuje pouze 2 typy kolizních objektů. Prvním typem objektu je kolizní úsečka. Ta představuje nepohyblivou překážku v prostoru. Druhým typem je kruhový objekt. Tento objekt se může volně pohybovat a může na něj působit vnější zrychlení, přičemž reaguje na kolize s kolizními úsečkami. Jako reakce na kolizi je použita nedokonale pružná srážka. Mezi kolizními objekty není žádné tření. Toto má za následek, že reakční impulz síly prochází vždy středem kruhu a není třeba počítat se setrvačným momentem. Simulace je prováděna v krátkých časových krocích v rámci kterých se vyhodnotí případné kolize a vypočítá nová poloha a rychlost.

4.2 Načtení kolizních úseček ze souboru SVG

Pro tvorbu a načtení kolizních objektů (složených z úseček), bylo využito možnosti získat atributy SVG elementů skrze javaScriptové API. Celá scéna byla vytvořena ve vektorovém editoru inkscape s tím, že kolizní elementy (představující kolizní objekty) byly uloženy do zvláštní vrstvy. Všechny tyto kolizní elementy byly vytvořeny jako objekty typu path, skládající se pouze z úseček. Vrstva v inkscapu představuje v rámci SVG element, jehož potomci jsou elementy v dané vrstvě. V SVG souboru byl tento element doplněn o id, což umožnilo následně k němu přistupovat z JavaScriptu pomocí funkce getElementById (součást DOM API). Pomocí dalších funkcí byly poté postupně získány jednotlivé kolizní elementy. Z těchto elementů byly získány souřadnice počátečních a koncových bodů kolizních úseček. Tyto souřadnice byly uloženy do pole. Dále byly předpočítány koeficienty obecných rovnic přímek na nichž tyto úsečky leží (dále testovací přímky). Výpočet je proveden tak, že se nejdříve z krajních bodů vypočtou nenormalizované koeficienty a_t a b_t. Tyto jsou poté vydělené délkou vektoru jež dohromady tvoří. Toto má za následek zjednodušení výpočtů při výpočtu kolizí. K takto vypočítaným koeficientům a a b je dopočítán koeficient c. Všechny tyto koeficienty jsou poté uloženy do pole.

Obecná rovnice přímky

$$a \cdot x + b \cdot y + c = 0 \tag{4.1}$$

Koeficienty a_t a b_t v nenormalizovaném tvaru

$$a_t = y_2 - y_1 \tag{4.2}$$
$$b_t = x_1 - x_2$$

Délka normálového vektoru přímky tvořeného koeficienty a_t a b_t

$$l = \sqrt{a_t^2 + b_t^2} \tag{4.3}$$

Normalizované koeficienty a a b (tvoří jednotkový vektor)

$$a = \frac{a_t}{l} \tag{4.4}$$
$$b = \frac{b_t}{l}$$

Konstanta c dopočítaná k normalizovaným koeficientům a a b

$$c = -a \cdot x_1 - b \cdot x_2 \qquad (4.5)$$

4.3 Nalezení kolize

Poté je podle rovnice (4.6) spočítána znaménková (nejde o absolutní hodnotu) vzdálenost středu kruhu od postupně všech testovacích přímek na nichž leží kolizní úsečky. (Toto je možné, protože se uvažuje malé množství kolizních úseček. V případě jejich vyššího počtu by bylo vhodné použít nějaký algoritmus na omezení počtu těchto úseček, testovaných na kolizi v jednom časovém kroku, např. BSP Tree nebo Quad Tree, v závislosti na povaze scény.) Rovnice (4.6) je zjednodušena oproti obvyklému tvaru, protože vektor tvořený konstantami *a* a *b* je jednotkový (viz výše).

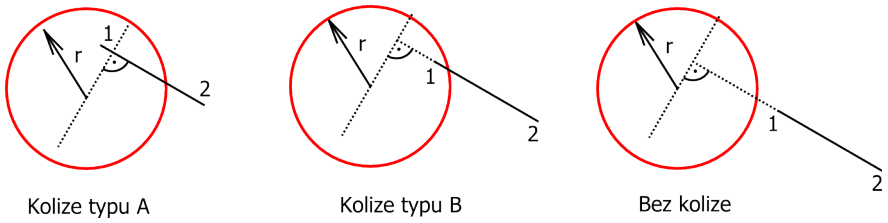
Znaménková vzdálenost středu kruhu a testovací

$$d = a \cdot x_k + b \cdot y_k + c \qquad (4.6)$$

kde x_k a y_k jsou souřadnice středu kruhového objektu a koeficienty *a*, *b* a *c* představují koeficienty obecné rovnice testovací přímky

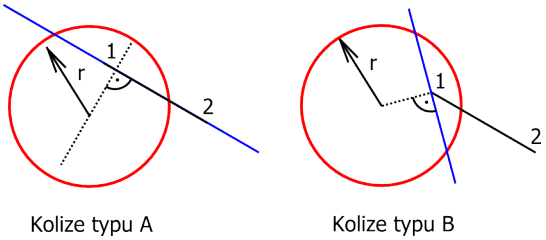
První podmínka kolize je že $|d|$ je menší než poloměr kruhového objektu *r*. Pokud je tato podmínka splněna je určena obecná rovnice přímky kolmé na testovací přímku. Konstanty *a* a *b* tedy odpovídají konstantám *-b* a *a* obecné rovnice testovací přímky. Konstanta *c* je dopočítána podle rovnice (4.5) tak, aby tato přímka procházela středem kruhu. Ze soustavy této rovnice a obecné rovnice testovací přímky je vypočítán průsečík. Porovnáním souřadnic tohoto průsečíku a souřadnic krajních bodů kolizní úsečky 3 různé stavy (viz Obr. 15):

- kolize typu A – průsečík leží mezi krajními body kolizní úsečky,
- kolize typu B – nenastala kolize typu A, ale vzdálenost středu kruhu a některého krajního bodu kolizní úsečky je menší než poloměr kruhu,
- bez kolize – nenastala ani kolize typu A ani kolize typu B.



Obr. 15 Tři možné stavy a jejich vyhodnocení kolizním algoritmem

V případě kolize se scéna zjednoduší na kolizi kolizní přímky a a kruhu. Pokud je kolize kolizí typu A kolizní úsečka se jednoduše nahradí kolizní přímkou (tj. přímkou na které tato úsečka leží). V případě kolize typu B (tj. kolize s krajním bodem kolizní úsečky) se kolizní úsečka nahradí přímkou procházející příslušným krajním bodem úsečky a je kolmá na spojnici středu kruhu s tímto krajním bodem. Nahrazení spočívá ve spočítání koeficientů obecné rovnice této přímky. Způsob vytváření náhradní přímky je vidět na Obr. 16 . Jde o určité zjednodušení které spoléhá na dostatečně malý časový krok. (viz dále) Jinak by totiž bylo nutné vypočítat souřadnici středu kružnice ve chvíli, kdy došlo ke kontaktu obou těles a na základě této provézt nahrazení kolizní přímkou.



Obr. 16 Náhradní kolizní přímka pro kolize typu A a B (modře)

4.4 Reakce na kolizi

Jak již bylo řečeno výše kolize je řešena jako nedokonale pružná srážka. Srážka se řídí rovnicí (4.7). V této rovnici vystupuje koeficient restituce, který nabývá hodnot v rozmezí 0 až 1 a vyjadřuje míru „pružnosti“ srážky. V případě krajních hodnot je o dokonale pružnou a dokonale nepružnou srážku. Protože kolizní úsečky představují pevně uchycené přepážky je možné jejich hmotnost uvažovat jako blížící se nekonečnu a rychlost rovnou nule. Na základě toho lze tuto rovnici zjednodušit na tvar (4.8).

Rovnice pro nedokonale pružnou srážku dvou těles [15]

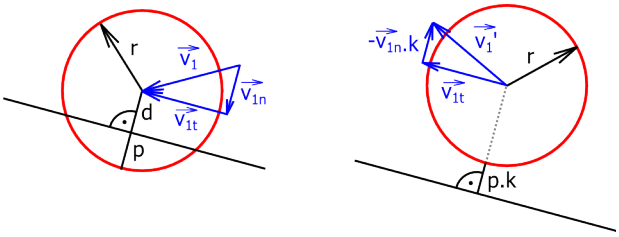
$$v_1' = \frac{(m_1 - k \cdot m_2) \cdot v_1 + (1+k) \cdot m_2 \cdot v_2}{m_1 + m_2} \tag{4.7}$$

kde v_1 je rychlost tělesa 1 před srážkou, v_2 je rychlost tělesa 2 před srážkou, m_1 je hmotnost tělesa 1, m_2 je hmotnost tělesa 2 a v_1' je rychlost tělesa 1 po srážce

Zjednodušení rovnice pro nedokonale pružnou srážku pro: $v_2 = 0$ a $m_2 \to \infty$:

$$v_1' = \lim_{m_2 \to \infty} \frac{(m_1 - k \cdot m_2) \cdot v_1 + (1+k) \cdot m_2 \cdot v_2}{m_1 + m_2} = \frac{(0 - k \cdot 1) \cdot v_1 + (1+k) \cdot 1 \cdot 0}{0+1} = -k \cdot v_1 \tag{4.8}$$

Protože jde po náhradě (viz výše) o kolizi přímky a kruhu mezi kterými není tření, týká se rovnice pro nedokonale pružnou srážku pouze normálové složky rychlosti vzhledem ke kolizní přímce. Následují rovnice pro výpočet rychlosti a polohy po srážce. Zobrazení poměrů před srážkou a po srážce lze vidět na Obr. 17 .



Obr. 17 Reakce na kolizi s kolizní přímkou

Průnik

$$p = (r - |d|) \cdot sgn(d) \tag{4.9}$$

kde r je poloměr kruhového tělesa a sgn je funkce jež nabývá hodnoty 1 pro kladné hodnoty a -1 pro záporné hodnoty

Normála kolizní přímky

$$\vec{n} = (a, b) \tag{4.10}$$

a, b jsou koeficienty obecné rovnice kolizní přímky

Korekce rychlosti

$$\vec{v_{kor}} = -(\vec{v_1} \cdot \vec{n} \cdot (1+k)) \cdot \vec{n} \tag{4.11}$$

Korekce polohy

$$\vec{x_{kor}} = -p \cdot (1+k) \,.\, \vec{n} \tag{4.12}$$

Nová rychlost

$$\vec{v_1}' = \vec{v_1} + \vec{v_{kor}} \tag{4.13}$$

Nová poloha

$$\vec{x}_1' = \vec{x}_1 + \vec{x_{kor}} \qquad (4.14)$$

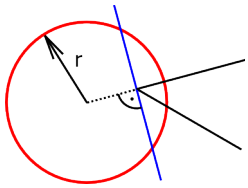
4.5 Vícenásobná kolize

Dalším problémem, který bylo nutné řešit, nastane, pokud se kruhový objekt dostane v časovém kroku do kolize s více objekty. Vícenásobné kolize lze rozdělit na 2 typy.

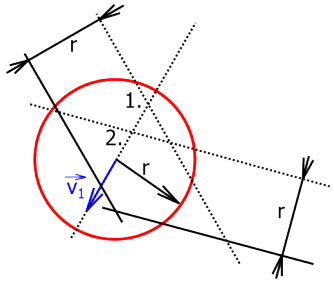
Prvním typem jsou vícenásobné kolize, kde však kolizní úsečky sdílí stejnou kolizní přímku. Toto jsou například rohy, tedy kolize s bodem, který je koncovým bodem více kolizních úseček zároveň. Použitý algoritmus tyto stavy detekuje a považuje je za jedinou kolizi (viz Obr. 18).

Druhým typem vícenásobné kolize je kolize s úsečkami, které ale nesdílí stejnou kolizní přímku. Takovéto kolize jsou uvažovány jako kolize, které nastaly v příliš krátkém časovém sledu, aby mohly být rozlišeny v rámci jednoho časového kroku. Algoritmus toto řeší tak, že s pomocí vektoru rychlosti a rovnoběžek s kolizními přímkami, určí pořadí ve kterém kolize nastaly. Poté je v tomto pořadí řeší (rozdělí problém na dva podproblémy, které dokáže řešit). Postup stanovení pořadí srážek je vidět na Obr. 19 .

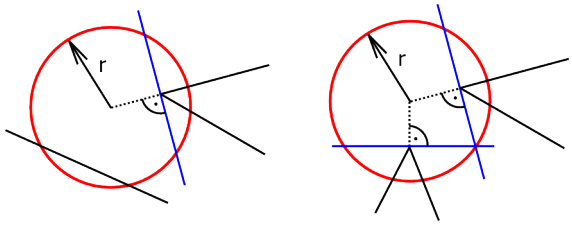
Počet těchto kolizí je však v aplikaci omezen na 2, to znamená, že hledání dalších kolizí se zastaví po nalezení 2 různých kolizních přímek (s využitím slučování popsaného výše). Toto vychází z předpokladu, že ve správně navrženém prostředí (pro tuto aplikaci) by kolize s více než dvěma kolizními přímkami nastat neměla (příklady různých srážek viz Obr. 20). I pokud by se simulace např. vlivem špatné počáteční polohy kruhového objektu do takovéto situace dostala, není často jasné jak by se měl algoritmus zachovat.(viz Obr. 21)



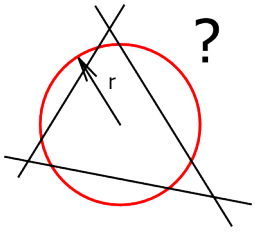
Obr. 18 Kolize s dvěma objekty sdílejícími stejnou kolizní přímku



Obr. 19 Využití vektoru rychlosti pro stanovení pořadí řešení kolizí s kolizními přímkami



Obr. 20 Kolize s více objekty tvořícími 2 kolizní přímky



Obr. 21 Neřešitelný stav kolize s třemi kolizními přímkami

4.6 Známé limitace algoritmu

Z výše uvedených zjednodušení plynou i určité limitace. Protože samotná detekce kolize vychází z předpokladu, že na konci časového kroku je objekt v kolizním stavu, může při nedostatečně malém kroku nebo příliš velké rychlosti kruhového objektu nezachytit některé „těsné“ kolize. Dále vlivem zjednodušení při tvorbě kolizní přímky, která se v případě kolize typu B (kolize s krajním bodem úsečky) konstruuje z polohy středu objektu na konci časového kroku, se úhel odrazu může mírně odlišovat od teoretické dráhy. Tato chyba se zvyšuje se zvětšujícím se časovým krokem a zvětšující se rychlostí objektu (závisí však i na dalších okolnostech). Toto je třeba brát v potaz při tvorbě světa a při volbě délky časového kroku (ta by se mohla např. upravovat v závislosti na rychlosti objektu).

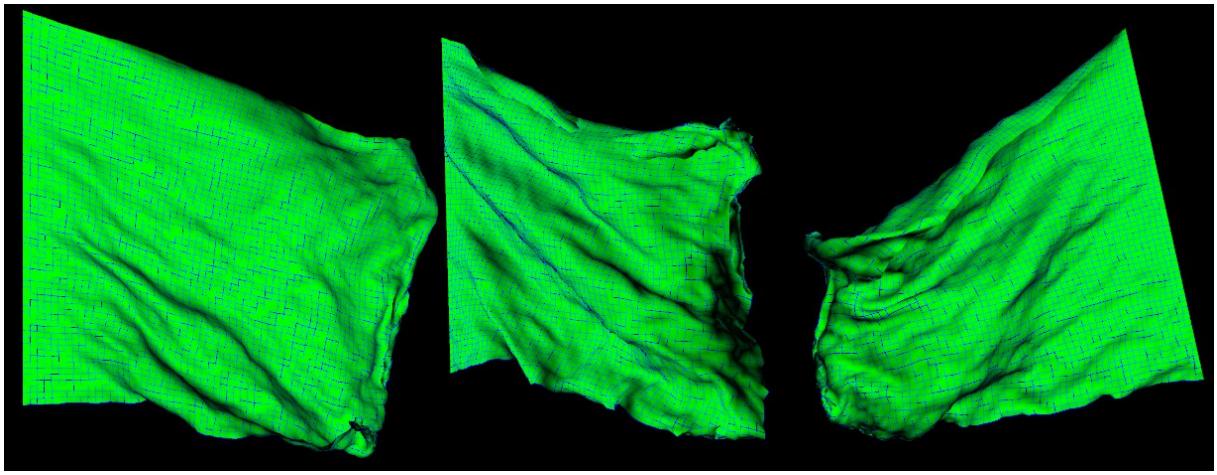
V případě, že by tato zjednodušení byla neakceptovatelná, bylo by možné toto řešit stanovením přesného okamžiku prvního doteku při tvorbě kolizní přímky, vyřešením srážky a poté opakováním celého algoritmu s případným řešením všech dalších kolizí v rámci časového kroku. To však za cenu vyšší složitosti i výpočetní náročnosti algoritmu.

5 ZÁVĚR

V první kapitole provedeno shrnutí webových technologií použitelných pro webové hry. Byly zkoumány možnosti a omezení jednotlivých technologií, z kterých byl poté stanoven výčet výhod a nevýhod jednotlivých technologií. Tímto byl splněn první cíl práce.

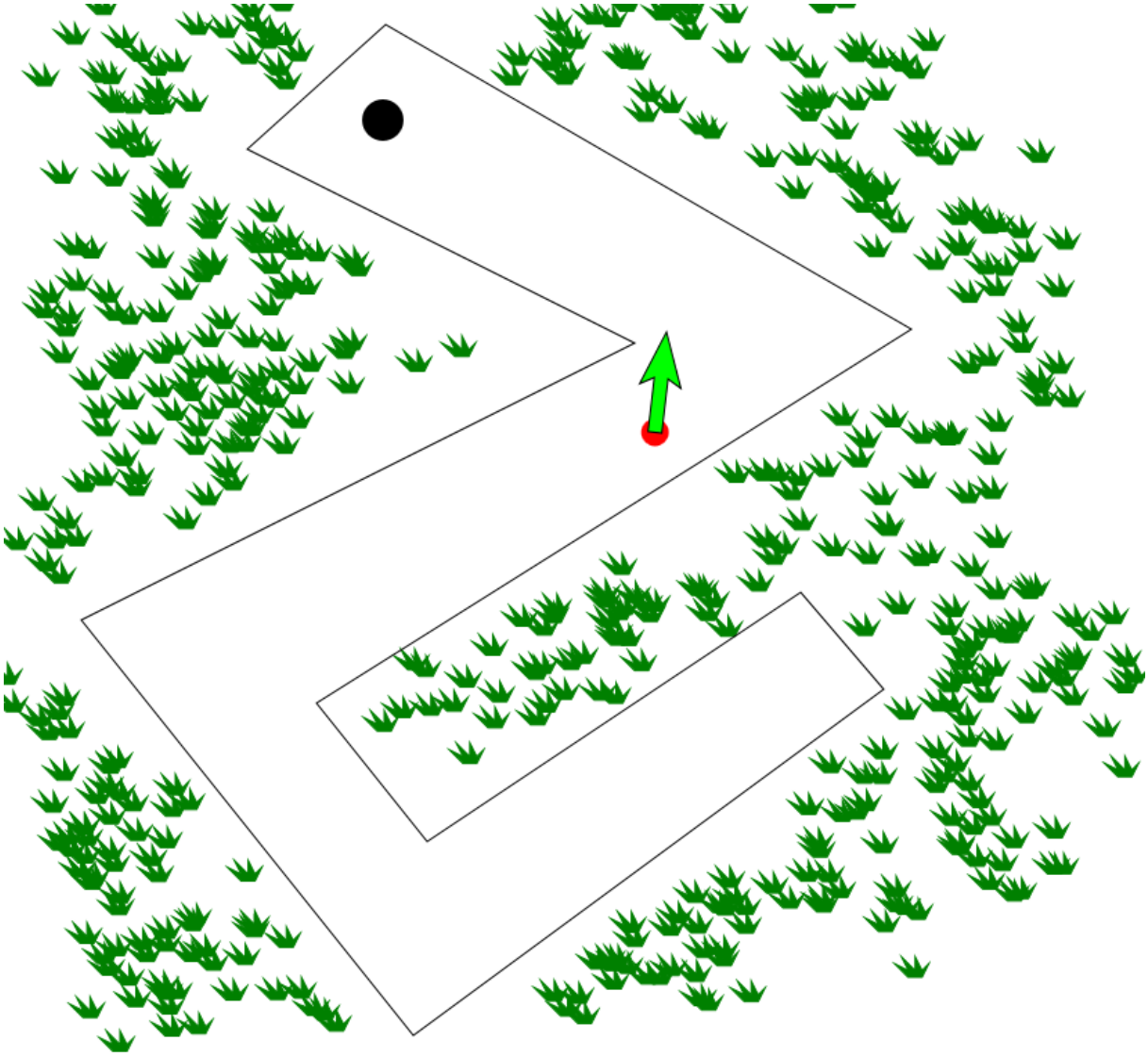
Dále byly pomocí technologií vytvořeny ukázkové aplikace, kterým jsou věnovány další 2 kapitoly.

První aplikace pojmenovaná „Vlajka ve větru“ byla vytvořena na demonstraci možností WebGL při tvorbě webových her. Cílem aplikace bylo numericky simulovat vlajku ve větru v reálném čase pro vizuální účely. Na základě těchto požadavků byl vytvořen matematický model. Protože jde o simulaci pouze pro vizuální účely, která však má být prováděna v reálném čase, byla uvažována určitá zjednodušení. Tato zjednodušení se týkala především působení větru. Dalším cílem bylo provádět výpočty na grafické a tím dosáhnout dostatečného výpočetního výkonu k provádění simulace v reálném čase. Celý problém byl tedy s pomocí WebGL mapován na grafickou kartu. Toto bylo provedeno tak, že data (vektorová pole) byla uložena do textur a výpočty prováděly shadery napsané pro tento účel renderováním do textury. Aplikace tedy splnila požadované cíle. Náhledy z aplikace lze vidět na Obr. 22 .



Obr. 22 Náhledy z aplikace „vlajka ve větru“

Druhá aplikace pojmenovaná „Aplikace simulující jednoduchou 2d fyziku“ byla vytvořena pomocí vektorového formátu SVG. Cílem aplikace bylo vytvořit jednoduchý 2D fyzikální engine, přičemž veškerá data scény, včetně kolizních překážek, načíst ze souboru SVG. Nejdříve byl proto s pomocí vektorového editoru Inkscape stanoven způsob jak tato data uvnitř souboru strukturovat. Následně byl v JavaScriptu napsán kód, k extrahování potřebných dat ze souboru. K tomu bylo využito DOM API s rozšířeními pro formát SVG, které webových prohlížeče nabízí. Dále byl vytvořen fyzikální model, který je v aplikaci použit. Tento model uvažuje pouze 2 typy objektů. První je kruhový objekt který se volně pohybuje v prostoru, druhým jsou pak kolizní úsečky, které jsou nepohyblivé. Kruhový objekt reaguje na kolizi s kolizními úsečkami nedokonale pružnou srážkou. Na základě tohoto modelu byla poté vytvořena samotná aplikace. I zde byla uvažována určitá zjednodušení, jejichž následky byly poté shrnuty na konci kapitoly věnované této aplikaci. Aplikace samotná se skládá z fyzikálního enginu, několika scén v SVG, které sloužily k testování algoritmu a jednoduché hry na bázi minigolfu (viz Obr. 23). I zde byly tedy splněny požadované cíle.



Obr. 23 Jednoduchá hra na bázi minigolfu

SEZNAM POUŽITÉ LITERATURY

[1] ECMAScript Language Specification. Ecma International [online]. 2011 [cit. 2013-05-22]. Dostupné z: http://www.ecma-international.org/ecma-262/5.1/Ecma-262.pdf

[2] SMOLA, Martin. HTML5: co přináší a proč se o něj zajímat. [online]. 29. 8. 2012. [cit. 2013-05-22]. Dostupné z: http://www.root.cz/clanky/html5-co-prinasi-a-proc-se-o-nej-zajimat/

[3] ORAM, John. Firefox 4 Arrives: Is it Any Good?. [online]. 2011 [cit. 2013-05-22]. Dostupné z: http://www.brightsideofnews.com/news/2011/3/22/firefox-4-arrives-is-it-any-good.aspx

[4] JAMES, John. Syncing, simplifying, and speeding up with Chrome's new beta. [online]. 2010 [cit. 2013-05-22]. Dostupné z: http://chrome.blogspot.cz/2010/08/syncing-simplifying-and-speeding-up.html

[5] Scalable Vector Graphics (SVG) 1.1 (Second Edition).W3C [online]. 2011 [cit. 2013-05-22]. Dostupné z: http://www.w3.org/TR/SVG/

[6] HTML5. W3C [online]. 2012 [cit. 2013-05-22]. Dostupné z: http://www.w3.org/TR/html5/

[7] *FAQ* [online]. 2013 [cit. 2013]. Dostupné z: http://wiki.whatwg.org/wiki/FAQ

[8] OpenGL ES 2.0 for the Web. Khronos Group [online]. 2013. vyd. [cit. 2013-05-22]. Dostupné z: http://www.khronos.org/webgl/

[9] WebGL Specification. Khronos Group [online]. 2013. vyd. [cit. 2013-05-22]. Dostupné z: https://www.khronos.org/registry/webgl/specs/1.0/

[10] MALIAROV, Michal. Netflix hodlá přejít ze Silverlightu na HTML5. *Živě* [online]. 2013. vyd. [cit. 2013-05-22]. Dostupné z: http://www.zive.cz/bleskovky/netflix-hodla-prejit-ze-silverlightu-na-html5/sc-4-a-168475/default.aspx

[11] WALLACE, Evan. *WebGL Path Tracing* [online]. 2010. vyd. [cit. 2013-05-22]. Dostupné z: http://madebyevan.com/webgl-path-tracing/

[12] WALLACE, Evan. *WebGL Water* [online]. 2011 [cit. 2013-05-22]. Dostupné z: http://madebyevan.com/webgl-water/

[13] MELAX, Stan. SOA Cloth Simulation with 256-bit Intel® Advanced Vector Extensions (Intel® AVX). [online]. 2012 [cit. 2013-05-22]. Dostupné z: http://software.intel.com/en-us/articles/soa-cloth-simulation-with-256-bit-intel-advanced-vector-extensions-intel-avx

[14] Numerical Integration [online]. 2012 [cit. 2013-05-23]. Dostupné z: http://wiki.vdrift.net/index.php?title=Numerical_Integration

[15] JÍRŮ, Josef. *Hybnost a energie při vzájemném působení těles* [online]. 2012 [cit. 2013-05-23]. Dostupné z: http://fyzikalniolympiada.cz/texty/hybnost.pdf

[16] WEISSTEIN, Eric W. *Runge-Kutta Method* [online]. 2013 [cit. 2013-05-23]. Dostupné z: http://mathworld.wolfram.com/Runge-KuttaMethod.html

[17] GORKIN, Travis. *Physically Based Cloth Simulation* [online]. 2009 [cit. 2013-05-23]. Dostupné z: http://www.gorkin.com/Cloth/cloth.html

[18] ČERMÁK, Libor a Rudolf HLAVIČKA. *Numerické metody* [online]. 2006 [cit. 2013-05-23]. Dostupné z: http://mathonline.fme.vutbr.cz/UploadedFiles/242.pdf